Australian Synchrotron

Galil EPICS driver configuration guide

Document no: 1

Revision no: 1.1

Date: 06 June 2016

Supported by

Australian Government

State Government Victoria

Synchrotron Light Source Australia Pty Ltd ABN 18 159 468 256 ACN 159 468 256 www.synchrotron.org.au
800 Blackburn Road, CLAYTON, VIC, 3168 **p** +613 8540 4100 **f** +613 8540 4200

REVISION HISTORY

Revision	Date	Prepared by	Description
1.0	27 April 2016	Mark Clift	Original
1.1	06 June 2016	Mark Clift	Added logging
1.2	27 June 2016	Mark Clift	Updated for driver version 3-3

TABLE OF CONTENTS

1 Building

1.1 Compiler

This software requires a compiler that is compliant with the C++11 standard because it makes use of the c++ 2011 structure “unordered-map”.

For linux, gcc versions 4.8.1 and greater are recommended. Some versions of linux have older gcc versions. In this case, the devtoolset2 chain or newer is recommended.

On windows, visual studio 2010 and above is required.

1.2 EPICS base

EPICS base version 3.14.12.2 or newer is required.

1.3 Required support modules

Module	Minimum version
ASYN	4-26
Autosave	5-5
SNCSEQ	2-1-8
SSCAN	2-8-1
CALC	3-4-2
Busy	1-6
Motor	6-8-1
IPAC	2-11

1.4 Release configuration

The Galil EPICS driver has both a config directory, and standard configure directory. The config/GALILRELEASE file is both a standard release, and a master release file. The configure/RELEASE file simply includes the GALILRELEASE file. The config directory supports building the Galil EPICS driver, and the included example IOC, using the SynApps build approach. To define the path to all required support modules, either;

1. Populate config/GALILRELEASE as a normal release file
2. Populate configure/RELEASE as a normal release file, and delete the following line include $(TOP)/config/GALILRELEASE

2 IOC Configuration

2.1 General principles

The Galil EPICS driver is highly integrated with EPICS, and most of the configuration steps are common with many EPICS applications found in the community. All settings are available at the EPICS database layer, and are accessible to the user through the provided displays. User levels are provided to protect the more sensitive settings from unwanted alterations. Autosave is used to save all the settings entered by the user at run time, and then to restore those settings after system start.

To configure the features of this driver usually three steps are required, they are:

1. Configure Autosave
2. Configure the database
3. Configure the IOC start scripts

It's recommended that the configuration engineer change only the controller prefix in the record names. To identify individual motors or ports, the record description field should be used instead. Record description fields are shown clearly on all of the screens, and are save/restored by autosave.

Given the above principles, this configuration guide does not assist with hardware configuration. Instead, supported hardware settings are shown on the provided displays, and these settings are then restored by autosave at IOC start. For example, the motor type, the encoder type for an axis is configured on the motor extras screen since these settings aren't supported by the motor record.

2.2 Create an IOC

Initially, create an IOC application using the EPICS makeBaseApp.pl script as shown below:

mkdir <IOCName>
cd <IOCName>
makeBaseApp.pl -t ioc <IOCName>
makeBaseApp.pl -i -t ioc <IOCName>

2.3 Adding Galil support

Initially, the Galil EPICS driver support module should be added to the IOC Makefile. To add support for the Galil EPICS driver, the following lines should appear in <IOCApp>/src/Makefile

```
<IOCName>_DBD += GalilSupport.dbd
<IOCName>_LIBS += GalilSupport
```

2.4 Autosave

Request files

Generally two autosave request files should be created in the iocBoot directory for an IOC. First, an autosave request file should be created to save the settings for all connected motor controller(s) and their associated motors. Second, an autosave request file should be created to save the motor positions. So, if two controllers are required on an IOC, the following autosave request files should be created in the iocBoot directory.

IOCName_settings.req
IOCName_positions.req

Configuration file

To configure the autosave utility, an autosave command script should be added to the iocBoot directory (eg. autosave.cmd), and to the IOC start script. So that autosave can find the request files that come with the Galil driver, autosave.cmd must contain the following lines:

```
set_requestfile_path("${GALIL}/GalilSup/Db", "")
set_requestfile_path("${MOTOR}/motorApp/Db", "")
set_requestfile_path("${TOP}/iocBoot/${IOC}", "")
```

To restore the correct autosave backup's at IOC initialization, the following lines should be added to autosave.cmd

```
# restore positions at pass 0 so motors don't move
set_pass0_restoreFile("IOCName_positions.sav")
# restore settings at pass 0
set_pass0_restoreFile("IOCName_settings.sav")
```

Also, the location where autosave will store the backups can be specified by adding the following line to autosave.cmd.

```
set_savefile_path("/autosave", "")
```

Refer to /iocBoot/iocGalil/autosave.cmd as an example autosave configuration file.

IOC start script

To create the autosave backup files whilst the IOC is in operation, a call to create_monitor_set should be added for each IOC autosave request file. The call to create_monitor_set should be placed after iocInit().

For example, to add two controllers, the following lines should be added to the IOC start script.

```
# Save motor positions every 5 seconds
create_monitor_set("IOCName_positions.req", 5,"P1=DMC01:, P2=DMC02:")
# Save motor settings every 30 seconds
create_monitor_set("IOCName_settings.req", 30, "P1=DMC01:, P2=DMC02:")
```

2.5 Controller

For Galil RIO installations only the “start script” section below is relevant, the autosave and database sections can be skipped.

Autosave

To save attributes that relate to the controller such as the limit switch type, and the “deferred moves” mode, the provided galil_ctrl_extras.req autosave request file can be used. The following line should be added to IOCName_settings for each controller:

file “galil_ctrl_extras.req” P=$(P)

For example, to add two controllers to IOCName_settings.req, add the following lines:

file “galil_ctrl_extras.req” P=$(P1)
file “galil_ctrl_extras.req” P=$(P2)

Database

The controller database should be loaded in the IOC start script using the provided galil_ctrl_extras.substitutions file. A single line entry should appear in galil_ctrl_extras.substitutions for each controller added to the system. To add two controllers, the following lines would appear in galil_ctrl_extras.substitutions.

```
# P    - PV prefix
# PORT - Asyn port name
# SCAN - Scan period for monitor records.  Use passive when only epics will change value (default)
#      - Periodic scan will slow controller update performance (poller)
# DEFAULT_LIMITTYPE = 0 Normally open, 1 Normally closed
# DEFAULT_HOMETYPE = 0 Normally open, 1 Normally closed
# PREC - precision

file "$(GALIL)/GalilSup/Db/galil_ctrl_extras.template"
{
pattern
{  P,          PORT,     SCAN,       DEFAULT_HOMETYPE, DEFAULT_LIMITTYPE, PREC}
 { "DMC01", "Galil1",    "Passive",  1,                1,                 5 }
 { "DMC02", "Galil2",    "Passive",  1,                1,                 5 }
}
```

Start script

GalilCreateController

The shell command GalilCreateController must be called for each controller in the IOC start script. The GalilCreateController command should be called before all other Galil EPICS driver commands for a particular controller. The GalilCreateController command has the following signature.

```
GalilController(const char *portName,
                const char *address,
                double updatePeriod)
```

The parameters are:

1. Const char *portName	- The name of the asyn port that will be created for this controller
2. Const char *address	- The address of the controller (eg. COM1, 192.6.94.5)
3. double updatePeriod	- The time in ms between datarecords 2ms minimum. Async if controller + bus supports it, otherwise is polled/synchronous. - Specify negative updatePeriod < 0 to force synchronous tcp poll period. Otherwise will try async udp mode first

For example to instantiate two controllers the following lines should be added to the IOC start script.

```
GalilCreateController("Galil1", "192.168.0.67", 8)
GalilCreateController("Galil2", "192.168.0.68", 8)
```

GalilStartController

The IOC shell command GalilStartController must be called for each controller in the IOC start up script. Also, the GalilStartController command must appear after all other Galil EPICS driver commands. In the same time, GalilStartController should appear before iocInit().

The GalilStartController command has the following specification

```
GalilStartController(const char *portName,
                     const char *code_file,
                     int burn_program,
                     int display_code,
                     unsigned thread_mask)
```

The parameters are:

1. Const char *portName	- The name of the asyn port for this controller
2. Const char *code_file	- The code file to deliver to controller. Usually “” to use generated code
3. double burn_program	- Should the code file be burnt to the controller EEPROM
4. int Thread mask	- Check these threads are running after controller code start. Bit 0 = thread 0 and so on. If thread mask = 0 and GalilCreateAxis appears > 0 then threads 0 to number of GalilCreateAxis is checked (good when using the generated code)

For example to start two controllers the following lines should be added to the IOC start script.

```
GalilStartController("Galil1", "", 1, 0)
GalilStartController("Galil2", "", 1, 0)
```

2.6 Custom code

The code that runs on the motor controller defines the homing behaviour, behaviour on limit switch activation, motor enable/inhibit interlock for each motor. In most cases, the Galil code generated by the Galil EPICS driver is sufficient, so parameter 2 for GalilStartController is most often just an empty

string “”. However, at times it may be desirable to change the code delivered to the controller. The code delivered can be altered by specifying a code file in parameter 2 of GalilStartController. When specifying a code file to use, GalilStartController can accept a single file, or it can use templates too. To specify the code using templates the following format should be used; headerfile;bodyfile1!bodyfile2!bodyfileN;footerfile

The Galil EPICS driver will write a copy of the code delivered to the controller in the iocBoot directory with extension .gmc.

To customize the controller code, the recommended process is:

1. Run the IOC once with no code file specified in GalilStartController
2. Open the controller code that has been dumped in iocBoot with an editor
3. Add custom code in the correct thread, but don’t change the structure
4. Save the file with a meaningful name
5. Specify the custom code file in GalilStartController

2.7 Analog IO

Autosave

For analog IO autosave is used to save/restore the user specified port descriptions. For analog outputs, the output voltage is also save/restored. To save the analog input settings, the provided `galil_analog_in.req` autosave request file can be used. To save the analog output settings `galil_analog_out.req` can be used.

For each analog input the following line should be added to IOCName_settings.req.

```
file "galil_analog_in.req" P=$(P), R=GalilAi0
```

Where P is the controller and R is the record name of the analog input. For example to save the settings for the first 2 analog inputs on 2 separate controllers, the following lines would be added.

```
file "galil_analog_in.req" P=$(P1), R=GalilAi0
file "galil_analog_in.req" P=$(P1), R=GalilAi1

file "galil_analog_in.req" P=$(P2), R=GalilAi0
file "galil_analog_in.req" P=$(P2), R=GalilAi1
```

For each analog output, the following line should be added to IOCName_settings.req.

```
file "galil_analog_out.req" P=$(P), R=GalilAo0
```

For example to save the settings for the first 2 analog outputs on 2 separate controllers, the following lines would be added.

```
file "galil_analog_out.req" P=$(P1), R=GalilAo0
file "galil_analog_out.req" P=$(P1), R=GalilAo1

file "galil_analog_out.req" P=$(P2), R=GalilAo0
file "galil_analog_out.req" P=$(P2), R=GalilAo1
```

Database

The analog IO database should be loaded in the IOC start script using the provided galil_analog_ports.substitutions file.

The galil_analog_ports.substitutions file is separated into input ports, and output ports.

To add an input or output port, a single line entry is required. For example to add the first 2 analog inputs, and first 2 outputs for 2 separate controllers, galil_analog_ports.substitutions would appear as below:

```
# P    - PV prefix
# R    - Record Name
# PORT - Asyn port name
# ADDR - Hardware port to read
# PREC - Precision

file "$(GALIL)/GalilSup/Db/galil_analog_in.template"
{
  pattern { P,          R,          PORT,    ADDR,  SCAN,      PREC }

#DMC Ports numbered 0 to 7 at database layer for GUI.
#DMC Ports numbered 1 to 8 on controller hardware
          { "DMC01",    "GalilAi0",  "Galil1", "1",   "I/O Intr", "3"  }
          { "DMC01",    "GalilAi1",  "Galil1", "2",   "I/O Intr", "3"  }

          { "DMC02",    "GalilAi0",  "Galil2", "1",   "I/O Intr", "3"  }
          { "DMC02",    "GalilAi1",  "Galil2", "2",   "I/O Intr", "3"  }

}

file "$(GALIL)/GalilSup/Db/galil_analog_out.template"
{
  pattern { P,          R,          PORT,    ADDR,  PREC,  LOPR,  HOPR }

#DMC Ports numbered 0 to 7 at database layer for GUI.
#DMC Ports numbered 1 to 8 on controller hardware
          { "DMC01",    "GalilAo0",  "Galil1", "1",   "3",   "-10", "10" }
          { "DMC01",    "GalilAo1",  "Galil1", "2",   "3",   "-10", "10" }

          { "DMC02",    "GalilAo0",  "Galil2", "1",   "3",   "-10", "10" }
          { "DMC02",    "GalilAo1",  "Galil2", "2",   "3",   "-10", "10" }

}
```

2.8 Digital IO

On digital motion controllers (DMC), and on the RIO PLC units, the digital inputs are organized in bytes. On DMC controllers the digital outputs are arranged in 16 bit words. On the RIO units the digital outputs are arranged in bytes. However, at the database layer the RIO digital outputs are named in words to be consistent with DMC units, so the same GUI can be used.

Autosave

For digital IO, autosave is used to save/restore the port descriptions. Also, for digital outputs, the output value is save/restored too. To save the digital input settings, the provided `galil_digital_in.req` autosave request file can be used. To save the digital output settings `galil_digital_out.req` can be used.

The digital inputs are arranged in bytes. So, for each digital input the following line should be added to IOCName_settings.req.

```
file "galil_digital_in.req" P=$(P), R=Galil0Bi0

Where;
0 = Byte number
0 = Bit number
```

Where P is the controller and R is the record name of the digital input. For example to save the settings for the first 2 digital inputs on 2 separate controllers, the following lines would be added.

```
file "galil_digital_in.req" P=$(P1), R=Galil0Bi0
file "galil_digital_in.req" P=$(P1), R=Galil0Bi1

file "galil_digital_in.req" P=$(P2), R=Galil0Bi0
file "galil_digital_in.req" P=$(P2), R=Galil0Bi1
```

The digital outputs are arranged in 16 bit words. So, for each digital output the following line should be added to IOCName_settings.req.

```
file "galil_digital_out.req" P=$(P), R=Galil0Bo0

Where;
0 = Word number
0 = Bit number
```

To save the settings for the first 2 digital outputs on 2 separate controllers, the following lines would be added.

```
file "galil_digital_out.req" P=$(P1), R=Galil0Bo0
file "galil_digital_out.req" P=$(P1), R=Galil0Bo1

file "galil_digital_out.req" P=$(P2), R=Galil0Bo0
file "galil_digital_out.req" P=$(P2), R=Galil0Bo1
```

Database

The digital IO database should be loaded in the IOC start script using the provided galil_digital_ports.substitutions file.

The galil_digital_ports.substitutions file is separated into input ports, and output ports.

To add an input or output port, a single line entry is required. For example to add the first 2 digital inputs, and the first 2 outputs for 2 separate controllers, galil_digital_ports.substitutions would appear as below:

```
# P    - PV prefix
# R    - Record Name
# PORT - Asyn port name
# BYTE - Hardware byte to read
# MASK - Mask for this bit

file "$(GALIL)/GalilSup/Db/galil_digital_in_bit.template"
{
pattern {P,      R,         PORT,  BYTE,  MASK,     ZNAM,  ONAM,  ZSV,        OSV        }

# DMC = Digital motor controller
# DMC binary inputs are organized in bytes
        {DMC01,  Galil0Bi0,  Galil, 0,     0x000001, "Off", "On",  "NO_ALARM", "NO_ALARM"  }
        {DMC01,  Galil0Bi1,  Galil, 0,     0x000002, "Off", "On",  "NO_ALARM", "NO_ALARM"  }
}

# P    - PV prefix
# R    - Record Name
# PORT - Asyn port name
# WORD - Hardware word to read
# MASK - Mask for this bit

file "$(GALIL)/GalilSup/Db/galil_digital_out_bit.template"
```

```
{
pattern {P,     R,          PORT,  WORD,  MASK,     ZNAM,  ONAM,  ZSV,       OSV       }

# DMC binary outputs are organized in 16bit words
        {DMC01,  Galil0Bo0,   Galil, 0,     0x000001, "Off", "On",  "NO_ALARM", "NO_ALARM"  }
        {DMC01,  Galil0Bo1,   Galil, 0,     0x000002, "Off", "On",  "NO_ALARM", "NO_ALARM"  }
}
```

2.9 Motor configuration

For each motor controller there are up to 8 real motors, and 8 coordinate system (CS) motors. The real motors are axis A thru H, and the CS motors are I thru P. The motor record addresses are numbered from 0. So the addresses of the real axes are 0 thru 7, and the CS axes addresses are 8 thru 15.

Autosave

Autosave is used to save and restore the motor record settings for real motors, and coordinate system motors (CSAxis). To save the motor record settings the motor_settings.req request file that comes with the motor record can be used.

For each motor the following line should be added to IOCName_settings.req.

File “motor_settings.req” P=$(P), M=A

Where;
A = The motor

For example to save the settings for the first 2 real motors, on 2 separate controllers, the following lines would be added to IOCName_settings.req.

File “motor_settings.req” P=$(P1), M=A
File “motor_settings.req” P=$(P1), M=B

File “motor_settings.req” P=$(P2), M=A
File “motor_settings.req” P=$(P2), M=B

Motor positions are also saved and restored by autosave. CS motors readbacks are calculated from real motor positions. For each motor, the following line should be added to IOCName_positions.req

```
$(P)A.DVAL
```

Where;
A = The motor

For example, to save the positions for the first 2 motors, on 2 separate controllers, the following lines would be added to IOCName_positions.req.

```
$(P1)A.DVAL
$(P1)B.DVAL
$(P2)A.DVAL
$(P2)B.DVAL
```

Database

The motor database should be loaded in the IOC start script using the provided galil_motors.substitutions file. At minimum, the galil_motor.template file should be used. Loading the motor records using the templates that come with the EPICS motor record is not supported by this driver. If the motor record templates from the motor record are used, the driver will not work.

For each motor, a single line entry should be placed in galil_motors.substitutions. Refer to GalilTestApp/Db/galil_motors.substitutions as an example.

Start script

For each real motor a call to GalilCreateAxis should appear in the IOC start script. GalilCreateAxis should be called after GalilCreateController, and before GalilStartController. The GalilCreateAxis function has the following specification:

```
GalilCreateAxis(const char *portName,
                char *axisname,
                int limit_as_home,
                char *enables_string,
                int switch_type)
```

The parameters are:

1. char *portName Asyn port for controller
2. char axis A-H
3. int limits_as_home (0 off 1 on) Use limits as home switch
4. char *Motor interlock digital port number 1 to 8 eg. "1,2,4". 1st 8 bits are supported
5. int Interlock switch type 0 normally open, all other values is normally closed interlock switch type

For example to instantiate the first 2 real motors for 2 separate controllers, the following lines would be added to the IOC start script.

```
GalilCreateAxis("Galil1","A",1,"",1)
GalilCreateAxis("Galil1","B",1,"",1)

GalilCreateAxis("Galil2","A",1,"",1)
GalilCreateAxis("Galil2","B",1,"",1)
```

2.10 Motor extra features

Motor extras are features specific to individual axis, and are not supported by the EPICS motor record. Many important attributes reside in motor extras such as; motor type, encoder type, home type definition, and so on. It is mandatory to setup motor extras for the real motors. Motor extras are not required or supported for CS motors.

Autosave

Autosave is used to save and restore the motor extra settings for real motors. To save the motor extra settings the galil_motor_extras.req request file that comes with the Galil EPICS driver can be used.

For each real motor the following line should be added to IOCName_settings.req.

```
file "galil_motor_extras.req" P=$(P), M=A
```

Where;
A = real motor
For example to add the first 2 motors, from 2 separate controllers, the following lines should be added to IOCName_settings.req

```
file "galil_motor_extras.req" P=$(P1), M=A
file "galil_motor_extras.req" P=$(P1), M=B

file "galil_motor_extras.req" P=$(P2), M=A
file "galil_motor_extras.req" P=$(P2), M=B
```

Database

The motor extras database should be loaded in the IOC start script using the provided galil_motors_extras.substitutions file.

For each real motor, a single line entry should be placed in galil_motor_extras.substitutions. For example to instantiate the first 2 motors, for 2 separate controllers galil_motor_extras.substitutions should appear as below:

```
# P - Controller
# M - Motor name
# PORT - Asyn port of controller
# ADDR - Axis number 0-7
# PREC - Precision of analog records
# SCAN - Scan period for monitor records.  Use passive when only epics will change value (default)
#          - Periodic scan will slow controller update performance (poller)
# MTRTYPE - motor type =
#                  0 - Servo
#                  1 - Reverse servo
#                  2 - High active stepper
#                  3 - Low active stepper
#                  4 - Reverse high active stepper
#                  5 - Reverse low active stepper
# MTRON - motor off 0, motor on 1

file "$(GALIL)/GalilSup/Db/galil_motor_extras.template"
{
pattern
{  P,      M,    PORT,     ADDR, PREC, SCAN,       MTRTYPE, MTRON, EGU  }
 {  "DMC01", "A",  "Galil1", 0,    3,    "Passive",  "3",     "0",   "mm" }
 {  "DMC01", "B",  "Galil1", 1,    3,    "Passive",  "3",     "0",   "mm" }

 {  "DMC02", "A",  "Galil2", 0,    3,    "Passive",  "3",     "0",   "mm" }
 {  "DMC02", "B",  "Galil2", 1,    3,    "Passive",  "3",     "0",   "mm" }
}
```

2.11 Coordinate System (CS) Motors

Since CS motors make use of the EPICS motor record, the autosave, and database setup procedure for CS motors is identical to that for real motors. However, CS motor positions are not managed by autosave. Further, motor extra settings are not relevant for CS motors.

Autosave

Refer to 2.9 Motor configuration.

Database

Refer to 2.9 Motor configuration.

Start script

The iocShell command GalilCreateCSAxes should be called in the IOC start script to create the CS axes. The call to GalilCreateCSAxes is generally placed after all GalilCreateAxis calls, and before the GalilStartController command. GalilCreateCSAxes creates all CS Axis I thru P at once. The GalilCreateCSAxes function has the following parameters:

```
GalilCreateCSAxes(const char *portName)
```

The parameters are:

1. char *portName Asyn port for controller

For example to create all CS axes for 2 separate controllers, the following should be added to the IOC start script.

```
GalilCreateCSAxes("Galil1")
GalilCreateCSAxes("Galil2")
```

2.12 Coordinate systems

Galil controllers provide 2 hardware coordinate systems known as the S and T planes. In contrast to other motor control systems, a coordinate system is only required when coordinating the motion of 2 or more motors in linear interpolation mode. Linear interpolation is used in “linear mode” profile motion, and “Sync start and stop” deferred moves. For example, in PVT mode > 2 motors can be coordinated without the use of a coordinate system S or T. For these reasons, the coordinate system planes are used under some circumstances only. Further, it’s not mandatory to configure the coordinate systems, even if using linear interpolation mode. Though configuring the coordinate systems is recommended for completeness.

Autosave

Autosave is not used or required for the coordinate systems.

Database

The coordinate system database should be loaded in the IOC start script using the provided galil_coordinate_systems.substitutions file.

For each controller, two lines should appear in galil_coordinate_systems.substitutions to instantiate the S and T coordinate systems. To instantiate the coordinate system for 2 controllers, galil_coordinate_systems.substitutions should appear as below:

```
# Coordinate system status
#
# P    - PV prefix
# R    - Record Name
# PORT - Asyn port name
# ADDR - Hardware port to read
# SCAN - Scan period for monitor records

file "$(GALIL)/GalilSup/Db/galil_coordinate_system.template"
{
pattern { P,         R,    PORT,    ADDR,  SCAN        }

      { "DMC01",    "S",  "Galil1", "0",   ".1 second"  }
      { "DMC01",    "T",  "Galil1", "1",   ".1 second"  }

      { "DMC02",    "S",  "Galil2", "0",   ".1 second"  }
      { "DMC02",    "T",  "Galil2", "1",   ".1 second"  }
}
```

2.13 Kinematics

Kinematics define the relationship between the real motors, and the CS motors. To use CS motors, it is mandatory to configure the kinematics. For a single CS motor there is 1 forward transform (readback), and 8 reverse transforms (setpoints). The kinematic variables meanwhile have controller wide scope. The kinematic equations and variables are runtime adjustable via the EPICS database.

Autosave

Autosave is used to save and restore the kinematics. Three things in the kinematics are save/restored by autosave, these are; forward kinematics, reverse kinematics, and kinematic variables. These items can be saved by using the provided galil_forward_transform.req, galil_reverse_transforms.req, and galil_kinematic_variable.req request files respectively.

To autosave the forward and reverse transforms for a single CS motor, the following lines would be added to IOCName_settings.req

```
file "galil_forward_transform.req" P=$(P), M=I
file "galil_reverse_transforms.req" P=$(P), M=I
```

Where;
I = CS motor

There are 10 kinematic variables Q thru Z that are available for use in kinematic equations. To autosave a kinematic variable, a single line entry should appear in IOCName_settings.req for each variable. For example to autosave the Q variable, the following line would appear in IOCName_settings.req.

```
file "galil_kinematic_variable.req" P=$(P), R=Q
```

To autosave the Q and R kinematic variables for 2 different controllers, the following lines would be added to IOCName_settings.req.

```
file "galil_kinematic_variable.req" P=$(P1), R=Q
file "galil_kinematic_variable.req" P=$(P1), R=R

file "galil_kinematic_variable.req" P=$(P2), R=Q
file "galil_kinematic_variable.req" P=$(P2), R=R
```

Where;
Q = Kinematic variable
R = Kinematic variable

Database

The kinematics for the CS motors should be loaded in the IOC start script using the provided galil_csmotor_kinematics.substitutions file. The substitution file is broken into different sections for forward transforms, reverse transforms, and kinematic variables.

To save/restore the forward kinematic equation for CS motor I the following would appear in the substitution file.

```
# Forward kinematic transform equations for CS motors
#
# 8 forward equations per controller.
# 1 forward equation per CS motor I-P (8-15) = 8 in total
# Eg. I=(A+B)/2 - Forward equations for CS motor I (8)
#
# P    - PV prefix
# M    - CSMotor name
# PORT - Asyn port name
# ADDR - CS Motor I-P (8-15)

file "$(GALIL)/GalilSup/Db/galil_forward_transform.template"
{
  pattern { P,          M,    PORT,    ADDR }
# CS motors (forward transforms)
```

```
        { "DMC01",     "I",  "Galil", "8"   }
}
```

To save/restore the reverse kinematic equations for CS motor I the following would appear in the substitution file.

```
# Reverse kinematic transform equations for CS motors
#
# 8 reverse equations per CS Motor that represent real motors A-H
# 64 reverse equations per controller in total as there are 8 CS motors I-P (8-15)
#
# Eg. A=I-J/2 - Reverse equation A for CS motor I (8)
#
# P    - PV prefix
# M    - CSMotor name
# PORT - Asyn port name
# ADDR - CS Motor I-P (8-15)

file "$(GALIL)/GalilSup/Db/galil_reverse_transforms.template"
{
  pattern { P,          M,    PORT,    ADDR }
# CS motors (reverse transforms)
        { "DMC01",     "I",  "Galil", "8"   }
}
```

To save/restore the first kinematic variable Q, the following would appear in the substitution file.

```
# Kinematic variables
#
# 10 variables per controller
#
# P    - PV prefix
# R    - Record Name
# PORT - Asyn port name
# ADDR - Hardware port to read
# PREC - Precision
#
# Regardless of record name, addr 0-9 is mapped Q-Z in driver
# These variables Q-Z can be used in kinematics

file "$(GALIL)/GalilSup/Db/galil_kinematic_variable.template"
{
  pattern { P,          R,    PORT,    ADDR,  PREC }

        { "DMC01",     "Q",  "Galil", "0",   "5"  }
}
```

An example substitution file is provided in GalilTestApp/Db/galil_csmotor_kinematics.substitutions.

2.14 Profile motion

Profile motion involves one or more motors following a user defined position profile. Profile motion can be executed using linear interpolation mode, or PVT mode. However, PVT mode is only available on the accelera series controllers and above. Some features of the profile motion relate to the controller (eg. profile mode, time base array), other features relate to the motors that will be used within the profile motion (eg. Absolute or relative move type).

Autosave

Autosave is not used with the profile motion settings.

Database

To load the profile features that relate to the controller the galil_profileMoveController.substitutions file should be loaded in the IOC start script. For an individual controller galil_profileMoveController.substitutions should appear as below;

```
# Profile move controller
#
# P    - PV prefix
# R    - Record Name
# PORT - Asyn port name
# NAXES – Number of axes in the profile
# NPOINTS – Number of profile points
#
file "$(MOTOR)/db/profileMoveController.template"
{
pattern
{P,      R,     PORT,   NAXES, NPOINTS, NPULSES,  TIMEOUT}
{DMC01:,  Prof1:, Galil,  8,     1441,    1441,     1}
}

# Profile move Galilcontroller
#
# P    - PV prefix
# R    - Record Name
# PORT - Asyn port name

file "$(GALIL)/GalilSup/Db/galil_profileMoveController.template"
{
pattern
{P,      R,     PORT,   TIMEOUT}
{DMC01:,  Prof1:, Galil,  1}
}
```

To load the profile features that relate to the axes, the galil_profileMoveAxes.substitutions file should be loaded in the IOC start script.

The provided file GalilTestApp/Db/galil_profileMoveAxis.substitutions illustrates how to populate galil_profileMoveAxes.substitutions for a single controller.

Start script

To use profile motion, the IOC shell command GalilCreateProfile must be called in the IOC start script. The GalilCreateProfile command should be called after all other Galil EPICS driver commands, except GalilStartController. The GalilCreateProfile command has the following signature.

GalilCreateProfile(const char *portName, int maxPoints)
The parameters are:

1. char *portName Asyn port for controller
2. Int maxPoints in trajectory

For example to use profile motion on 2 separate controllers, the following should be added to the IOC start script.

```
GalilCreateProfile("Galil1", 2000)
GalilCreateProfile("Galil2", 2000)
```

2.15 Logging

Solicited command/response traffic can be logged by the IOC. To enable logging, add the following line to the start script.

```
epicsEnvSet("GALIL_DEBUG_FILE", "galil_debug.txt")
```

When using the provided example IOC, uncomment the above line in iocBoot/iocGalil/st.cmd to enable logging.

3 User display configuration

MEDM, and QT displays are provided with the Galil EPICS driver. The QT screens are recommended as they provide a more intuitive experience, a modern look and feel, and “user levels” to protect the engineering settings from accidental change by non-engineering staff or users.

3.1 MEDM

The MEDM main screen for digital motor controllers can be started by using the provided start_dmc_screen.sh script. The interface for RIO PLC controllers can be started using the provided start_rio_screen.sh script.

The screen start scripts mentioned above use the config/GALILRELEASE file to construct the MEDM environment variable EPICS_DISPLAY_PATH. Using this environment variable, MEDM can find the necessary community screens, and the screens provided with this driver. It may be necessary to alter the screen start scripts to use configure/RELEASE instead of GALILRELEASE, depending on the release configuration decisions made in section 1.4

Finally, the line that invokes MEDM may need to be updated to ensure the correct macros are used.

3.2 QEGUI

A Qt, EPICS framework known as QEUI has been created at the Australian Synchrotron. The QEGUI framework is available for download here:
https://sourceforge.net/projects/epicsqt/

The QT main screen for controllers can be started using the provided start_qedmc_screen.sh script. While the RIO PLC interface can be started by using the provided start_qerio_screen.sh script.

The screen start scripts mentioned above use the config/GALILRELEASE file to construct the QEGUI environment variable QE_UI_PATH. Using this environment variable, QEGUI can find the necessary screens provided with this driver. It may be necessary to alter the screen start scripts to use configure/RELEASE instead of GALILRELEASE, depending on the release configuration decisions made in section 1.4

Finally, the line that invokes QEGUI may need to be updated to ensure the correct macros are used.